TECHNICAL REPORT 3010
May 2016

Convolutional Neural Network on Embedded Linux® System-on-Chip

A Methodology and Performance Benchmark

Daniel Gebhardt, Ph.D
Keyur Parikh
Iryna Dzieciuch

SSC Pacific
San Diego, CA 92152-5001

SSC Pacific
San Diego, California 92152-5001

K. J. Rothenhaus, CAPT, USN
Commanding Officer

C. A. Keeney
Executive Director

ADMINISTRATIVE INFORMATION

The work described in this report was performed by the IO Support to National Security Branch (Code 56120), the Mission Systems Engineering Branch (Code 56170), and the Environmental Sciences Branch (Code 71750), Space and Naval Warfare Systems Center Pacific (SSC Pacific), San Diego, CA. The Naval Innovative Science and Engineering (NISE) Program at SSC Pacific provided funding for this Applied Research project.

Released by
E. R. Buckland, Head
IO Support to National Security Branch

Under authority of
G. Settelmayer, Head
Information Operations Division

EXECUTIVE SUMMARY

Deep convolutional neural networks (CNNs) detect and classify features of interest in sensory input data. There is a need to investigate how best to implement CNNs for Navy and Department of Defense (DoD) use in platforms with minimal size, weight, and power (SWaP) capacity, since much academic research focuses solely on achieving the highest performance on a specific dataset with minimal concern of compute resources. What are the Pareto-optimal points and trade-offs for an energy-efficient CNN, when considering both its architecture and underlying implementation?

This report describes a methodology, configuration, and experimental results of a first step in this study—a baseline for comparison of benchmarking metrics. A baseline is important for quantifying any further results and to estimate potential benefits of new and more advanced ideas.

An embedded-system development platform (Xilinx® ZC702) was used to implement a CNN and benchmark its performance. The ZC702 completed the task of classifying a set of 10,000 handwritten digits in 13,281 ms, compared to a laptop-class computer, which took 841 ms. Additionally, another CNN configuration took 5007 ms, compared to 341 ms. These results can be summarized in that the ARM® Cortex®-A9 CPU in the ZC702 takes approximately 15 times longer than the Intel® i7-4770HQ CPU in the laptop. The estimated power usage of each is 1 W for the Cortex®-A9 CPU and 15 W for the i7 CPU.

A method of accelerating this computation is by using a customized hardware unit called a field-programmable gate array (FPGA). The ZC702 contains this logic within its main system-on-chip (SoC). A rough estimate of performance improvement is also presented in this report. The logic for neuron-computation was designed and simulated, with the results extrapolated to the larger CNN as run in the experiment. Based on this estimation, the FPGA accelerates the task such that it completes it in 7.5% of the time taken by the A9 CPU alone. There is no power estimate at this time, but it is estimated to reduce consumption by at least 50%, compared to the A9 CPU alone.

The recommended action for implementing low-SWaP CNNs (or other machine-learning techniques) is to accelerate the operation with FPGA hardware, or other specialized hardware components, if performance and SWaP are critical to optimize. A corollary to this is that if the task requires a relatively low computational throughput, a software-only implementation may be sufficient running directly on the embedded SoC platform.

CONTENTS

Figures

Tables

1. INTRODUCTION

Artificial neural networks, and specifically deep convolutional neural networks (CNNs), are a top-performing technology to detect and classify features of interest in sensory input data. The most common input data is imagery, audio, and text data, with the output providing a descriptive label of an image [1] or music [2], for example. Defense Advanced Research Project Agency (DARPA) and other Department of Defense (DoD) agencies have funded research in this field, and private industry has also heavily invested. Generally, CNNs and related machine learning approaches are a quickly growing and potentially disruptive technology in many application areas.

CNNs are commonly operated using laptop-sized to supercomputer-class equipment, including general-purpose graphical processing units (GPGPUs). Many potential Navy use-cases would require a compute system to have much lower size, weight, and power (SWaP) than traditional approaches offer, while maintaining a high-performance.

An example could be a watch-sized camera and processor attached to a small unmanned aerial vehicle that can identify people holding firearms from an overhead view. Or, consider a small, easily-installable device that detects malicious computer behavior (e.g., a cyber incident) by analyzing properties of network traffic and operating system function calls.

CNNs operate in two phases or modes: training and testing. The goal of training is to determine the optimal parameter values in the CNN (these are called synapse weights and neuron biases) such that the error is minimized performing a given task with a training-only dataset. In test mode, which we can also call *deployed* mode, the CNN is operating on data it has not seen before in training. The parameters values are fixed at this point, from the training phase, and are not altered with the new data samples. There are methods to update weights in an online fashion, but these details are outside the scope of this report. The training phase requires a significantly greater amount of computational power than the deployed phase. This is due to iteratively evaluating the network, adjusting its parameters, and repeating the process. While training offers an opportunity for accelerating its computation through GPGPUs, FPGAs, and high-performance computers, it does not necessarily help the problem of a deployed CNN in a low-SWaP platform. Thus, we focus this project's effort on deployed-mode operation.

1.1 PROJECT GOAL: LOW-SWaP CNN

Much academic research focuses solely on achieving the highest performance on a specific dataset with minimal concern of compute resources. The trend is for deeper and more complex networks to be used, consuming as much training time and resources as possible—often on the order of weeks or months—to eke out a few tenths of a percent accuracy performance.

The research question that follows is mostly unanswered: what are the Pareto-optimal points and trade-offs for an energy-efficient CNN, when considering both its architecture and underlying implementation? This report provides the supporting groundwork for this study, and specifically considers the *deployed* phase of operation towards the goal of enabling many important DoD-relevant solutions.

2. METHODOLOGY AND EXPERIMENT

This report documents a performance comparison between a laptop-class processor and an embedded-class processor operating a CNN in *deployed* mode.

Three platforms were evaluated: the laptop system, the embedded system, and the laptop system emulating the embedded system. Each platform ran the CNN as a software application utilizing its respective CPU. Additionally, a FPGA design was developed that implements the neuron calculation in customized hardware. This design was not yet integrated to the CNN design in a way that allows a direct comparison, but its simulated performance results are used to estimate the gains of a completed system.

2.1 LAPTOP PLATFORM

The laptop system was a 15 in. Apple® Macbook® with an Intel® i7-4770HQ CPU. This system does not have a general-purpose graphics processing unit (GPGPU). Development and evaluation of the CNN took place on a Linux® virtual machine running through Oracle® VirtualBox™. Since this platform contains commodity hardware and software, it is a good benchmarking baseline.

2.2 SYSTEM-ON-CHIP EMBEDDED PLATFORM

The embedded platform chosen for this work was the Xilinx® ZC702 development kit. This kit is a single board containing a Xilinx® Zynq®-7000 system-on-chip (SoC) and many interface peripherals, such as Universal Serial Bus (USB), Secure Digital (SD)-card, and Ethernet. The board is low power, and the SoC does not have a heat sink. The SoC chip is approximately 2 cm x 2 cm in dimensions and is low-power enough to not require a heat sink. Note that a final system could be made much smaller than this development board, which has "wasted" space compared to a board used in a finished product.

The Zynq®-7000 contains two CPU cores (ARM® Cortex®-A9) with double-precision floating-point support, and the Artix®-7 FPGA fabric for implementation of custom logic to accelerate computational workloads. This FPGA fabric, in addition to the standard programmable logic, contains 220 DSP (digital signal processing) "slices" that provide fast and energy-efficient mathematical operations. For this work, the CPU was configured to operate at 667 MHz, which is the default for the Xilinx®-provided reference design.

This SoC was configured to run a mile®-based operating system called Petalinux. A Linux® operating system is desirable to provide a familiar and flexible approach to interfacing with the SoC. This allow common applications to run such as `sshd` or `vi`, and easily enable devices such as a mouse, keyboard, ethernet, or video display.

2.3 LAPTOP EMULATION OF EMBEDDED SYSTEM

A common approach to developing software for an embedded system is to emulate that system through a virtual-machine. This work utilized the QEMU software to virtualize and emulate a Zynq-7000 SoC. The advantage of emulation is that the physical hardware is not needed to run the

application-under-development, which is especially useful during debugging. Emulation is typically much slower than running an application natively because the emulator must translate the machine instructions from one instruction set architecture (ISA) to that of the host ISA. Therefore, emulation may not provide accurate performance estimations of the target embedded system. Nevertheless, these results are presented to provide an interesting data point for future work.

2.4 CNN CONFIGURATION

A CNN configuration defines the architecture and architectural parameters of the network. Examples of these parameters include:

- Input data dimensions and channels (e.g., image size and colors)
- Size of convolutional filters
- Number of convolutional filters
- Pooling/downsampling size and method (e.g., max-pool or average)
- Number of convolution and pooling layers
- Size and number of fully connected (dense) layers
- Output representation size and type (e.g., the number of classes of the input dataset and the predicted class of an input sample)

The CNN configuration for this work was chosen such that it can be experimented with easily and quickly. This capability is important, since large CNNs are unwieldy to work with, and could hamper fast iteration of experimentation. For example, a 8-layer network took six days to train on the ImageNet dataset using two NVIDIA® GTX580 GPUs [3]. Therefore, for this initial work, we decided to concentrate on small networks and small datasets until the methods are matured.

2.5 CNN ARCHITECTURE

The CNN configuration was specified as follows, and visualized in Figure 1. The input was a one-dimensional series of 100 values that range between $[0, 1.0]$. This corresponds to a two-dimensional input image of 10x10 pixels.

The first convolutional layer uses a filter (also called a *neuron* or *kernel*) of size 3x3 pixels, a stride of 1, and 10 separate filters. Each filter corresponds to a set of weights (3x3 in this case) that are convolved across the image pixels in both the X and Y dimensions, producing an output value for every stride. The activation function of this layer is *rectified linear*, which simply enforces a lower bound of 0 on the input (no negative outputs); otherwise, it outputs the (positive) value at the input. Most recent CNN work utilizes this activation function over hyperbolic tangent or sigmoidal functions since it leads to more rapid training.

The next layer is a pooling layer, configured to average the activations within a 2x2 pixel window. It effectively subsamples the output, reducing each dimension's resolution by half.

The final layer is fully-connected to the pooling layer's output, producing N outputs, where N is the number of classes in the dataset. In this example, the number of classes is 10, for the digits 0 through 9, as discussed below. This fully connected layer (FCL), therefore, has 10 neurons, each with 160 inputs and 160 synapse weights. For a given image example at the CNN input layer, the output will most strongly activate the output neuron in this layer corresponding to the digit it perceives as the most likely classification, provided proper training has been completed to determine all neuron weights.

More details of these operations can be found in many textbooks and academic papers [4].

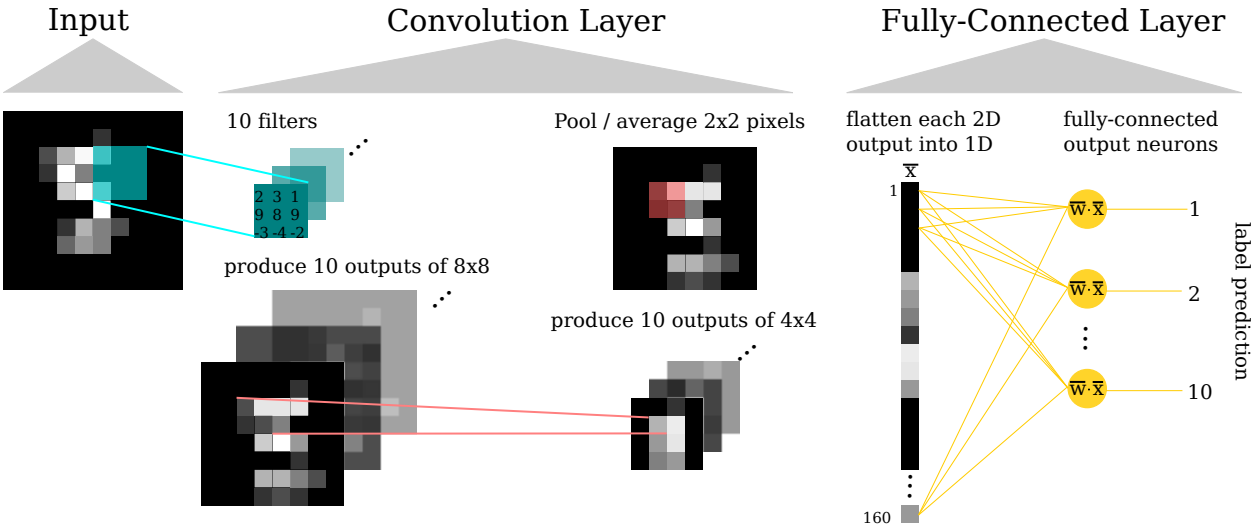

Figure 1. CNN architecture for custom MNIST dataset classification.

2.6 DATASET

The dataset chosen for this task is derived from the MNIST digit classification data [5]. We resized each image of the grayscale MNIST dataset to be 10x10 pixels, which was done with the OpenCV C++ library using the `INTER_CUBIC` method of resampling. The pixel values were normalized to the range $[0, 1.0]$. The resized images were stored in the database format LevelDB, which is a key-value file-based storage method. The format of these keys and values is as follows. Keys are a 8-byte unsigned integer representing the numerical index of a sample. The values are a concatenation of the 8-byte class label (the true label for a sample) and the pixel intensities (64-bit floating point value per pixel). Note, this pixel format is not the most space-efficient, since each pixel value can sufficiently be represented with an 8-bit integer (0–255) but the neural network calculations are done with floating-point values. As per normal procedure, the dataset is split into *test* and *train* partitions (of sizes 10,000 and 60,000 images, respectively) to avoid testing on already-observed data (*data-snooping*), which would artificially inflate the classification performance. Examples from this dataset are shown in Figure 2.

2.7 CNN FRAMEWORK

Over the past several years, several frameworks for developing, training, and testing CNNs have arisen. Examples of these frameworks include Keras, Cafe, MatConvNet, Tensorflow™, and others.

Figure 2. Sample digits from the resized 10x10 pixel MNIST dataset.

Tiny CNN is one such framework that is designed to require minimal external libraries and dependencies, as it relies solely on C++ standard libraries. This makes it ideal for embedded systems, where libraries are not often readily available. Thus, we have chosen Tiny CNN for this work. For the x86 platform, the GNU C++ compiler (GCC, g++) version 4.8.4 was used to build the CNN executable from the Tiny CNN framework. For the embedded ARM platform, a GCC-derived commercial compiler was used: Sourcery™ Codebench Lite v.4.9.2. This compiler is the included default for the Xilinx® tools.

2.8 CNN TRAINING AND DEPLOYMENT

Training the CNN was done on the laptop platform, and the trained network parameters (e.g., the neurons' weights and biases) were saved to a file. As mentioned previously, the *train* partition of the dataset was used to optimize the network parameters using stochastic gradient descent (SGD), with the *AdaGrad* adaptive gradient extension [6]. SGD is the most commonly used optimization approach for neural network training, and the AdaGrad approach reduces the need to search for an optimal learning rate manually. A batch size of 50 images was used, with 50 epochs (complete passes) through the training set.

The parameter file was then copied to the embedded platform, where it was used to instantiate the network in a ready-to-use state. This approach is an important aspect of neural networks: a single network architecture can be used for a variety of tasks by simply changing its parameters, which can be specified with an easy data file update. The algorithm and executable program do not necessarily have to change (although the input and output dimensions would need to match, or at least be adapted to match).

One approach for gaining insight to the trained network is to visualize the filters of the first convolutional layer, which is done by plotting the filter weights as a square image, corresponding to the pixel locations of the input. Figure 3 shows this plotting, where lighter colors represent higher weights. One can see some structure in the filters—notice the two containing light-colored 2x2 squares in the third and sixth column. These neurons are sensitive to an upper left and lower left corners. Or, consider the filter furthest right that is sensitive to a diagonal-like edge.

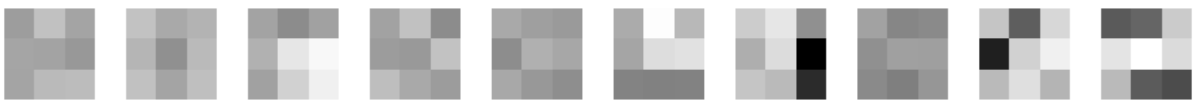

Figure 3. Visualization of neuron weights in the convolutional layer.

With such a small filter size (3x3), it is difficult to see the typical Gabor filters that are learned with larger filters and with a more diverse set of training data (e.g., all natural scenes) [3]. Also note that

L2 regularization was not used during training, which reduces the overall weight magnitude. For a simple network, this reduction is generally unnecessary, but it is a commonly used technique to help reduce overfitting and increase generalization of the learned features.

Deployment was done by copying the following items to the Petalinux filesystem: the CNN executable application, the custom MNIST dataset in LevelDB format, and the neuron parameter (weight and bias values) file. The Petalinux filesystem was then written to a SD memory card, creating a boot disk for the embedded device. Any future additions or modifications can be done through a variety of methods common to Linux® systems, including the use of SSH to copy files and access the embedded system, `tftp`, or even a network or Internet distribution method.

3. RESULTS

3.1 CNN TASK PERFORMANCE

While not directly important to the overall goal of this study, the CNN described above classified 92% of the test set accurately. No other published work on this resized-MNIST dataset exists, and this lack of research can certainly be improved by adding additional CNN resources. Qualitatively, it is a reasonable result, considering the limited size of the CNN and input space.

3.2 CPU COMPARISON

The time taken to run the whole *test* dataset partition through the CNN on each platform is shown in Table 1. The rows list the recorded metric (time) in milliseconds, and a fraction comparing the platform to the i7 baseline platform. The Zynq® platform takes 15.8 times longer for execution than the i7 platform.

Table 1. Platform execution time of CNN on resized MNIST dataset.

	Intel® i7-4770HQ	QEMU of Zynq®-7000	Zynq®-7000 Hardware
Time (ms)	841	8385	13281
Time vs. i7	1.0	10.0	15.8

A second experiment was done to validate the results of the first. This CNN architecture was slightly different: a single (1) filter of size 4x4 pixels, no bias values, and a "stride" of 2. The stride is the distance in pixels between two filter applications, such that a stride of 2 reduces the input size by a factor of two. The dataset used for this consists of randomly generated pixel values of 10x10 size, and is termed "synthetic", in that it doesn't represent anything particular, but allows for platform performance evaluation. 100,000 images were used for the test set. Results are listed in Table 2, and track closely with the first experiment. The slightly smaller performance difference between the Zynq® and i7 is hypothesized to be caused by the different architecture. The second experiment has fewer filters and no fully-connected layer, and thus iterates over the input images more often. This means there is proportionally more sequential code and memory access than arithmetic operations. The i7 has a compute architecture that is much better at out-of-order execution of instructions than the ARM® Cortex® A9 architecture. Therefore, the more computationally-intensive first-experiment favors better performance on the i7.

Table 2. Platform execution time of 2nd CNN architecture on synthetic dataset.

	Intel® i7-4770HQ	QEMU of Zynq®-7000	Zynq®-7000 Hardware
Time (ms)	341	4522	5007
Time vs. i7	1.0	13.3	14.7

3.3 FPGA ACCELERATION ESTIMATE

The Zynq®-7000 SoC includes a FPGA logic block to provide customizable hardware that can be used to implement computationally intensive pieces of an application in a more energy- and time-efficient method, compared to the CPU. The nature of an FPGA makes it especially good at parallel

operations or stream processing applications. This capability is in contrast to applications with heavy control-flow or branching operations, where a CPU may be more desirable. A CNN is an ideal use of FPGA logic, as it contains many parallelizable operations, consisting of mostly multiplies and additions.

Making use of the FPGA can prove a developmental challenge. An increasingly common method to move algorithms to the FPGA is the concept of *high-level synthesis*, which allows software to be written in common C or C++ languages, simulated and validated for correct operation, and then synthesized into the hardware description languages of Verilog or VHDL.

We used the Xilinx® development tool suite, Vivado®, to create a sample neuron layer, and evaluate its potential performance. This neuron layer contained five neurons, each connected to a 20-element input vector. Each neuron pipelines the vector multiply and sum operation of its 20-element inputs and weights. The layer parallelizes across neurons, so each of the five neurons executes in parallel. Vivado® estimated the computation interval as 192 cycles at 200 MHz, or in other words, an output vector is produced from one input vector every 192 cycles.

3.3.1 FPGA Compute Capacity

This neuron layer does not match up in size or configuration with the CNN architecture described above, but it can be used as a reference point to estimate the expected performance. First, the compute capacity of the whole FPGA is estimated. The layer above consumes 11% of the FPGA's resources, specifically the DSP slices and lookup tables (LUTs). This percentage means we can increase the number of neurons in the layer by eight times, from 5 to 40, and estimate the FPGA resources increase to 88%, allowing for overhead from other logic.

For each 20-element input vector (IV), the FPGA can process 800 multiplies and 200 additions per interval. We can refer to both of these arithmetic operations together as simply neuron *operations*. While not identical in computation, it is close enough for this estimation purpose. To find the neural operations per second:

$$\frac{800 + 200\ ops}{1\ IV} \times \frac{1\ IV}{192\ cycles} \times \frac{200\ cycles}{1\ \mu second} \times \frac{10^6\ \mu second}{1\ second} = 1.0 \times 10^9\ ops/second$$

3.3.2 CNN Compute Requirement

The 10 x 10 pixel MNIST classification requires the arithmetic operations per image shown in Table 3. Convolutional layers require vector multiplication and an adder-tree that sums the element-wise products. Pooling layers require comparisons between elements in the "pool", which is implemented as a subtract operation (which itself is similar to an addition).

Table 3. Number of required arithmetic operations for 10 x 10 MNIST CNN classification.

	Layer 1	Layer 2	Layer 3	Total
Mults.	5760	0	1440	7200
Adds or Subtracts	2560	320	72	2952
Grand Total				10152

3.3.3 FPGA Speedup

We can now answer the question of how many images per second the FPGA can process.

$$\text{FPGA image throughput: } \frac{10^9 ops}{1\ second} \times \frac{1\ image}{10152\ ops} = 98500\ images/sec$$

For comparison, the ARM CPU in the Zynq has the following throughput.

$$\text{ARM CPU image throughput: } \frac{10000\ images}{13.281\ seconds} = 753\ images/sec$$

The final estimated speedup is thus, $98500/753 = 131$ times faster.

This is very close to similar work that implemented CNN in a similar Zynq® SoC, sponsored by Office of Naval Research (ONR) [7], where the authors measured a speedup of 112 times for their FPGA vs. ARM® CPU comparison.

4. CONCLUSION AND RECOMMENDATIONS

4.1 ADVANTAGES OF EMBEDDED CNN

Numerous algorithms can be used for any machine learning task. Why is a neural network, and specifically a CNN desirable, especially for embedded systems? First, the computation requirements are "embarrassingly parallel," meaning that energy-efficient custom logic can be utilized for a massive advantage in the embedded space. Second, CNNs are very adaptable for different sensing domains. Imagery is often used, but others are equally suitable such as audio [8], text-based event detection and analysis [9], or language translation as Google, Inc. currently publicly operates [10]. Finally, the engineering development work invested in building such a system is potentially easier to reuse than other more domain-specific machine learning approaches. For example, the detection task of a embedded CNN can easily be changed from finding firearms to finding cars, simply by updating the parameters (weights) with a new file (that can be trained on an external high-performance computer).

4.2 RECOMMENDATIONS

While academia and industry are pursuing neural network technology with great resources, the DoD must also continue its focus on specific aspects that are relevant to its needs. This is especially true of embedded, low-SWaP methods that can be used in situations lacking powerful computers and reliable network connections. There are two major thrusts to this recommendation: novel techniques and novel domains.

4.2.1 Novel Techniques

The requirement of limited computational resources on a low-SWaP device should motivate additional research into techniques that can better use those resources. For example, how can the convolutional process efficiently identify calculations that won't greatly inform the end result, and simply stop the computation? Or, what regularization technique is most efficient for undersea acoustic monitors utilizing the limited power from seafloor bacteria? These are questions that can be answered with continued effort into novel and DoD-specific techniques.

4.2.2 Novel Domains

Adapting existing and new techniques to domains of interest to the Navy and DoD is a fruitful course of research. Adaptation includes defense-specific sensors and intelligence data focusing on specific needs (e.g., identify a particular underwater acoustic signature), and more general needs (e.g., draw conclusions from intelligence reports). A domain of particular interest to SSC Pacific is that of cyber-related data and analysis. A high-speed, hardware-accelerated CNN may provide an ideal platform for making intelligence from raw data as it appears from the network, operating system, or user experience. This idea was initially explored using other machine learning techniques at SSC Pacific, and published in multiple technical reports [11] and [12].

4.2.3 Dataset Development

A critical aspect to CNN research is that of the dataset, especially when working in novel domains. Additional work should be invested in to create datasets specific to DoD use-cases. The research and demonstrations that follow will be much more compelling than the ubiquitous "academic" datasets, lending credibility of these techniques to produce operationally relevant solutions.

REFERENCES

1. Vinyals, O., A. Toshev, S. Bengio, and D. Erhan. 2015. "Show and Tell: A Neural Image Caption Generator." The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 8–10 June, Boston, MA.

2. Humphrey, E. J., J. P. Bello, and Y. LeCun. 2012. "Moving Beyond Feature Design: Deep Architectures and Automatic Feature Learning in Music Informatics." *Proceedings of the International Society of Music Informationa Retreival (ISMIR)* (pp. 403–408). October 8–12, Porto, Portugal.

3. Krizhevsky, A., I. Sutskever, and G. E. Hinton. 2012. "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 25*:1106–1114.

4. Goodfellow, I., Y. Bengio, and A. Courville. 2016. "Deep Learning." Available online at http://goodfeli.github.io/dlbook/. Accessed April 14, 2016.

5. Lecun, Y., and C. Cortes. "The MNIST Database of Handwritten Digits." Available online at http://yann.lecun.com/exdb/mnist/. Accessed April 14, 2015.

6. Duchi, J., E. Hazan, and Y. Singer. 2010. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley. Available online at http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-24.html. Accessed April 14, 2016.

7. Gokhale, V. J. Jin, A. Dundar, B. Martini, and E. Culurciello. 2014. "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (696–701). June 23 and 28, Columbus, OH.

8. Montavon, G. 2009. "Deep Learning for Spoken Language Identification," NIPS Workshop on Deep Learning for Speech Recognition and Related Applications. December 12, Whistler, British Columbia, Canada. Available online at http://research.microsoft.com/en-us/um/people/dongyu/NIPS2009/. Accessed April 14, 2016.

9. Nguyen, T. H. and R. Irishman. 2015. "Event Detection and Domain Adaptation with Convolutional Neural Networks," *Proceedings of the Association for Computational Linguistics* (pp. 365–371). Available online at http://aclweb.org/anthology/P/P15/P15-2060.pdf . Accessed April 22, 2016.

10. Mykolaiv, T., Q. V. Le, and I. Sutskever. 2013. "Exploiting Similarities among Languages for Machine Translation," *Google, Inc. arrive preprint arXiv:1309.4168.*

11. Straatemeier, L., G. Palavincini, E. Dorman, and S. Melvin. 2015. "Behavioral-Based Peer-to-Peer Botnet Detection, with Exploration into Unlabeled Navy Network Data," SSC Pacific Technical Report 2096. Space and Naval Warfare Systems Center Pacific (SSC Pacific), San Diego, CA.

12. Parameswaran, S., S. Melvin, M. Crewes, and E. Dorman. 2014. "To Catch A Botnet." IEEE Military Communications Conference. October 6–8, Baltimore, MD.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-01-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
May 2016	Final	

4. TITLE AND SUBTITLE

Convolutional Neural Network on Embedded Linux® System-on-Chip
A Methodology and Performance Benchmark

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHORS

Daniel Gabhardt, Ph.D
Keyur Parikh
Iryna Dzieciuch

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

SSC Pacific, 53560 Hull Street, San Diego, CA 92152–5001

8. PERFORMING ORGANIZATION REPORT NUMBER

TR 3010

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

SSC Pacific
Naval Innovative Science and Engineering (NISE) Program
53560 Hull Street
San Diego, CA 92152–5001

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Deep convolutional neural networks (CNNs) detect and classify features of interest in sensory input data. There is a need to investigate how best to implement CNNs for Navy and Department of Defense (DoD) use in platforms with minimal size, weight, and power (SWaP) capacity, since much academic research focuses solely on achieving the highest performance on a specific dataset with minimal concern of compute resources.

This report describes a methodology, configuration, and experimental results of a first step in this study—a baseline for comparison of benchmarking metrics. A baseline is important for quantifying any further results and to estimate potential benefits of new and more advanced ideas.

15. SUBJECT TERMS

Mission area: Autonomy and Machine Intelligence
convolutional neural networks; low size, weight and power; system-on-chip; field-programmable gate array

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Daniel Gebhardt
U	U	U	U	18	**19B. TELEPHONE NUMBER** *(Include area code)* (619) 553-2786

INITIAL DISTRIBUTION

84300	Library	(2)
85300	Archive/Stock	(1)
56120	D. Gebhardt	(1)
56170	K. Parikh	(1)
71750	I. Dzieciuch	(1)

Defense Technical Information Center
Fort Belvoir, VA 22060–6218 (1)

Approved for public release.

SSC Pacific
San Diego, CA 92152-5001